NAME	Ms.NEERJA DOSHI
UID	2021300029
DIV	SE – CE -A
DAA EXPT 1 - B	
DATE OF PERFORMANCE	31 . 01 . 23

AIM :	**Experiment on finding the running time of an algorithm.**
ALGORITHM :	**1) SELECTION SORT :** Initialize minimum value(min_idx) to location 0. Traverse the array to find the minimum element in the array. While traversing if any element smaller than min_idx is found then swap both the values. Then, increment min_idx to point to the next element. Repeat until the array is sorted **2) INSERTION SORT :** Iterate from arr[1] to arr[N] over the array. Compare the current element (key) to its predecessor. If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

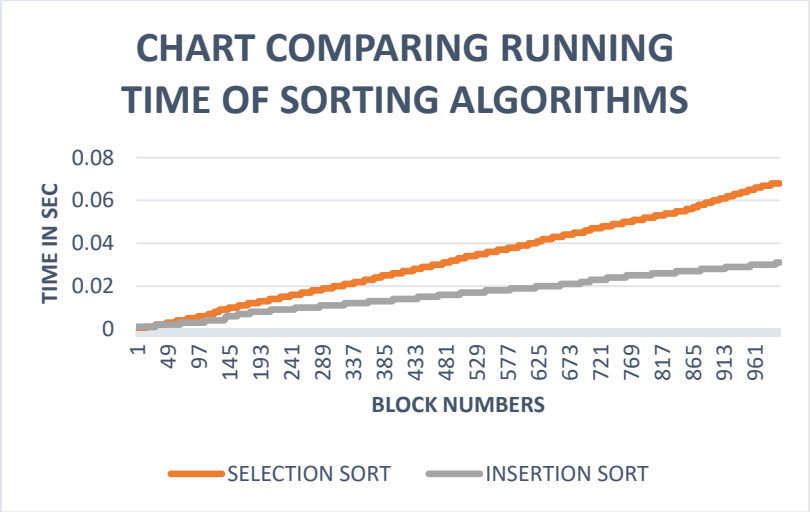
CODE :

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

double *selectionSort(int a[][100], int blocks, int nos);
double *insertionSort(int a[][100], int blocks, int nos);
void swap(int *xp, int *yp);
void printTime(double *timer_selection, double *timer_insertion,
int blocks);

int main()
{
   clock_t begin, end;

   // 50 rand -- 10 array of 5 elements each
   int blocks = 1000, nos = 100;
   // int blocks = 2, nos = 10;
   int arr[blocks][nos];
   int offset = 0;

   // for const -- not puting seed == time(Null)
   // srand(time(NULL));
   srand(0);

   // itme elapsed between generation + printing nos
   begin = clock();

   for (int i = 0; i < blocks; i++)
   {
      for (int j = 0; j < nos; j++)
      {
         arr[i][j] = rand() % 100 + offset;
      }
      offset += 100;
   }

   // print unsorted array of random numbers
   for (int i = 0; i < blocks; i++)
   {
      printf("BLOCK %d\t\t\n", (i + 1));
      for (int j = 0; j < nos; j++)
      {
         printf("%d\t", arr[i][j]);
      }
      printf("\n\n");
   }

// putting output in csv
   FILE *textfile;
   textfile = fopen("random_number_generater.csv", "w");
   for (int i = 0; i < blocks; i++)
```

```
    {
        for (int j = 0; j < nos; j++)
        {
            fprintf(textfile, "%d ", arr[i][j]);
        }
        fprintf(textfile, "\n\n");
    }

    fclose(textfile);

    end = clock();

    double time_to_generate_print = ((double)end - begin) /
CLOCKS_PER_SEC;

    printf("TIME ELAPSED IN GENEARTION AND PRINTING RANDOM
NUMBERS IS : %f\n\n", time_to_generate_print);

    // insertionSort(arr, blocks, nos);
    // double* timer_selection = selectionSort(arr, blocks, nos);
    // printTime(selectionSort(arr, blocks, nos) , insertionSort(arr,
blocks, nos) , blocks) ; intended
    // but time(insertion) < time(selection) hence changed

    printTime(insertionSort(arr, blocks, nos), selectionSort(arr,
blocks, nos), blocks);
    return 0;
}

// swapping for selection sort
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

double *selectionSort(int a[][100], int blocks, int nos)
{
    clock_t begin, end;
    begin = clock();                    // here for 1 to 2 , 1 to 3 , 1 to 4
... 1 to 10000 bloks
    double *timer = malloc(sizeof(double) * blocks); // keeps track of
time elapsed in bloks

    // printf("BLOCK \t\t TIME TO SELECTION SORT\n") ;

    // algo
    for (int k = 0; k < blocks; k++)
    {
        // begin = clock();  here for 1 ,  2 , 3 ,  4 ...  10000 individual
```

```
bloks

      // sorting within the interior array
      int i, j, min_idx;

      // One by one move boundary of unsorted subarray
      for (i = 0; i < nos - 1; i++)
      {
         // Find the minimum element in unsorted array
         min_idx = i;
         for (j = i + 1; j < nos - 1; j++)
            if (a[k][j] < a[k][min_idx])
               min_idx = j;

         // Swap the found minimum element with the first element
         if (min_idx != i)
            swap(&a[k][min_idx], &a[k][i]);
      }

      end = clock();
      double time_to_selection_sort = ((double)end - begin) /
CLOCKS_PER_SEC;

      // printf("1 TO %d\t\t %f\n", (k + 1), time_to_selection_sort);
      timer[k] = time_to_selection_sort;
   }

   return timer;
}

double *insertionSort(int a[][100], int blocks, int nos)
{
   clock_t begin, end;
   begin = clock();                        // here for 1 to 2 , 1 to 3 , 1 to 4
... 1 to 10000 bloks
   double *timer = malloc(sizeof(double) * blocks); // keeps track of
time elapsed in bloks

   // algo
   for (int k = 0; k < blocks; k++)
   {
      // begin = clock();  here for 1 ,  2 , 3 ,  4 ...  10000 individual
bloks

      // sorting within the interior array
      for (int j = 0; j < nos; j++)
      {
         int i, key, m;
         for (i = 1; i < nos; i++)
         {
            key = a[k][i];
```

```
            m = i - 1;

            while (m >= 0 && a[k][m] > key)
            {
               a[k][m + 1] = a[k][m];
               m = m - 1;
            }
            a[k][m + 1] = key;
         }
      }

      end = clock();
      double time_to_insertion_sort = ((double)end - begin) /
CLOCKS_PER_SEC;

      timer[k] = time_to_insertion_sort;

   }
   return timer;
}

void printTime(double *timer_selection, double *timer_insertion,
int blocks)
{

   printf("BLOCK \t\t TIME TO SELECTION SORT \t TIME TO
INSERTION SORT\n");

   for (int i = 0; i < blocks; i++)
   {
      printf("1 TO %d\t\t %f \t\t\t %f\n", (i + 1), timer_selection[i],
timer_insertion[i]);
   }

   free(timer_selection);
   free(timer_insertion);
}
```

GRAPHICAL REPRESENTATION :	
OBSERVTIONS :	1) INSERTION SORT IS FASTER THAN SELECTION SORT 2) SPACE COMPLEXITY FOR BOTH THE ALGORITHMS IS CONSTANT 3) BOTH GRAPHS INCREASE EXPONENTIALLY
CONCLUSION :	By conductiing the above experiment ive realized that insertion sort is a faster sorting algoritm than selection sort irrespective of the size of output when the time to run program of sorting is plotted on a 2d graph , the difference between both alogorithms can be evidently seen